CVE 2021-26814

Relazione sulla difesa dalla CVE 2021-26814

wazuh/**wazuh**

Wazuh - The Open Source Security Platform

Rabbito Paolo

X81000073
Università degli Studi di Catania
Internet Security
2020/21

SOMMARIO

INTRODUZIONE

Wazuh è un software gratuito ed open-source finalizzato alle funzioni di intrusion detection, intrusion prevention e azioni di risposta in caso di intrusioni. Le componenti del software sono 3: Wazuh agent, Wazuh server e Elastick stack.
Wazuh agent si occupa di fornire servizi di instrusion detection e di intrusion prevention nei client, inoltre raccoglie dati sul sistema e sulle applicazioni da inoltrare al servizio server attraverso un canale autenticato e criptato.
Wazuh server mette a disposizione un servizio di analisi dei dati che gli vengono inoltrati dagli agent, inoltre si occupa di attivare gli alert nel momento in cui vengono rilevate attività anomale. In questa relazione si analizzerà nello specifico ad analizzare il modulo RESTful API di questo componente, che si occupa di fornire un'interfaccia diretta per interagire con l'infrastruttura di Wazuh, permettendo di gestire gli agent collegati, cambiare le regole per rilevare intrusioni, modificare le impostazioni del server e per ottenere dati derivati dal monitoraggio del server.
Ed infine Elastick stack è una suite di progetti open source che vengono integrati da Wazuh .
Nella sua versione 4.0.0 è stata rilevata una vulnerabilità nel modulo RESTful API della sua componente server identificata dall'id CVE 2021-26814, quest'ultima è rimasta presente fino alla versione 4.0.3 del software. Nella seguente relazione si analizzerà e descriverà questa vulnerabilità, successivamente si andrà a vedere un possibile percorso logico che possa portare a trovare il suo exploit ed infine si analizzeranno le contromisure adottate. Si andrà anche a dimostrare il tutto con una demo testando l'exploit una prima volta per evidenziare la vulnerabilità e una seconda volta per verificare che le tecniche di difesa applicate siano valide. In particolare utilizzeremo la versione 4.0.3, ovvero l'ultima versione giudicata vulnerabile secondo la mitre corporation.

IPOTESI

Il modulo Wazuh API dalla versione 4.0.0 alla 4.0.3 di Wazuh consente ad un utente autenticato di eseguire codice arbitrario con privilegi di amministratore sul server che ospita il servizio attraverso l'URI /manager/files.
Un utente autenticato al servizio potrebbe sfruttare una non efficiente e sicura validazione dell'input per effettuare l'injection di codice arbitrario nello script eseguito

dal servizio API una volta effettuata una privilege escalation.

TESTBED UTILIZZATO

Per effettuare i test di utilizzeranno 3 macchine virtuali collegate tra loro in una rete interna. Le prime due macchine avranno installato due versioni differenti dell'applicativo per permettere di evidenziarne le differenze una volta aggiornato.
La terza macchina invece sarà la macchina attaccante, dalla quale si eseguirà uno script python nel tentativo di eseguire codice arbitrario con permessi di root nella macchina della vittima.

1. Macchina xubuntu 20.04 con Wazuh 4.0.3
2. Macchina xubuntu 20.04 con Wazuh 4.0.4
3. Macchina Kali per testare il servizio più vecchio e quello aggiornato

ANALISI GENERALE

Il modulo Wazuh API è una RESTful API che permette di interagire con il Wazuh manager in modo semplice. Attraverso questa interfaccia potremo aggiungere degli agent, richiedere dei dati o riavviare il manager. L'accesso alle funzioni messe a disposizione dall'interfaccia è regolamentato da delle policy di tipo RBAC(Role-based access control). Sarà quindi necessario autenticarsi e farsi riconoscere come un determinato utente con determinati permessi. Il sistema di autenticazione quindi richiede un username e una password e ci restituirà un token necessario per effettuare le successive richieste al servizio che risulterà possibile effettuare in base al profilo con cui ci autentichiamo e i permessi che ne conseguiranno. Inoltre risulta importante, ai fini della relazione, notare come nella sezione "Get files content" della documentazione relativa all'API viene spiegato che è possibile effettuare richieste GET, PUT e DELETE all'URI /manager/files, ottenendo in risposta il contenuto di qualsiasi file. Ovviamente con "qualsiasi file" la documentazione sottintende qualsiasi file accessibile secondo le policy definite.
Ora, queste richieste possono essere effettuate tramite una richiesta di questo tipo: "GET https://ipaddress:55000/manager/files?path=...".
La variabile path viene successivamente validata più volte, e nello specifico viene

sottoposta ad un controllo nelle funzioni: format_etc_ruleset_file_path (api/api/validator.py), get_file (framework/wazuh/manager.py) e validate_cdb_list (framework/wazuh/manager.py). Ma cosa succederebbe se potessimo eludere questi tre controlli e di conseguenza modificare e sovrascrivere dei file a cui non dovremmo avere accesso? E se inserissimo del codice malevolo all'interno di questi?

EXPLOIT

Per trovare un exploit alla nostra CVE partiremo analizzando le funzioni precedentemente elencate, inoltre seguiremo lo stesso ragionamento seguito da Davide Meacci sulla base del quale quest'ultimo ha realizzato uno script che permette di sfruttare la vulnerabilità a vantaggio dell'attaccante. L'autore dello script infatti non arriva direttamente ad una local escalation, bensì si trova ad analizzare e testare tutta una serie di attività che, come vedremo, forzano quelli che sono i controlli del nostro applicativo. Infatti, se è vero che la vulnerabilità in sé viene descritta in modo molto preciso come: "la possibilità di eseguire codice arbitrario con privilegi di amministratore attraverso uno specifico URI", è anche vero che analizzarla in un contesto pratico dove viene sfruttata ci permetterà di comprenderla meglio.
La funzione format_etc_ruleset_file_path si occupa di controllare se il path che gli viene passato in input come parametro sia un path valido e accessibile. Ovvero si occupa di controllare che sia concesso utilizzare quel path a seguito di una richiesta all'URI manager/files. Questo controllo viene effettuato prima verificando che il path sia valido per la funzione is_safe_path e successivamente che corrisponda ai criteri definiti dalla seguente regex `(^(etc|ruleset)\/(decoders|rules|lists(\/[\w\-\_]+)*))$.`

Dalla regex sappiamo che la variabile path dovrà iniziare con una concatenazione di caratteri del tipo "etc/lists" e dalla is_safe_path possiamo vedere che questa si occupa di concatenare attraverso l'operatore "." la stringa "var/ossec/" al nostro path.
Questo espone l'applicativo al path trasversal attack di cui abbiamo parlato prima, basterà infatti utilizzare un path del tipo: "/etc/lists/../ossec.conf" per rendere il path appetibile per la nostra funzione di check e comunque effettuare una richiesta valida per un percorso che non dovrebbe essere accessibile tramite l'API.
Tuttavia nella pratica, andando a guardare bene la funzione get_file scopriamo che, se anche riuscissimo a passare il controllo da parte della funzione is_safe_path, non

riusciremmo a passare il controllo dalla funzione get_file, poichè la concatenazione della nostra variabile path con “var/ossec/” questa volta avviene tramite la funzione join e non tramite l’operatore “.”.
Per ovviare a questa problematica sarà necessario tornare indietro fino alla root del path, di conseguenza il path che inseriremo nella nostra richiesta all’API diventerà qualcosa diAlmeno in teoria, perchè simile a “/etc/lists/../../../../../../var/ossec/etc/ossec.conf”.
Risulta interessante notare come adesso potremo effetturare una richiesta get all’URI “/etc/lists/../../../../../../var/ossec/api/configuration/security/jwt_secret”, ottenedo così la chiave che, qualora non avessimo i permessi per effettuare certe operazioni permetterebbe di generare un token valido. Non sarà questo il caso, ma trovo importante sottolinearlo in quanto questo garantisce che le policy RBAC dell’applicativo non saranno fonte di limitazione per questo tentativo di exploit.
Alla luce di quanto fino ad ora esposto è plausibile che, avendo abbastanza permessi, potremo leggere tutto ciò che si trova all’interno di /var/ossec/, tuttavia è ancora più interessante interessante che oltre ad una richiesta GET sia possibile effettuare una richiesta PUT.
Ovviamente aggirare questi controlli sulla variabile path è solamente una parte di ciò che serve per trovare un exploit alla nostra vulnerabilità. Infatti come anticipato si tratta di una vulnerabilità legata ad una privilege escalation locale che permetterebbe di ottenere accesso in scrittura a file ai quali non dovrebbe essere concesso l’accesso se non ad un utente root.
Dalla documentazione risulta che anche la richiesta PUT accetta una variabile path, la quale verrà validata, come visto prima per il metodo GET, con una piccola differenza nella regex utilizzata per il confronto, che questa volta sarà `(^etc\/(ossec\.conf|(rules|decoders)\/[\w\-\/]+\.xml|lists\/[\w\-\.\/]+)$)`.
Una volta validata la stringa path il software passerà al controllo del contenuto del file che si sta cercando di caricare. Per fare ciò il software prima creerà un file temporaneo, in cui verrà copiato il contenuto del file che si tenta di caricare, e successivamente, questo file temporaneo dovrà essere validato dalla funzione validate_cdb_list, che appunto si occuperà di controllare che il contenuto del file sia formattato come il software si aspetta, ovvero in formato di lista cdb.
Il formato cdb è generato da una lista di coppie “chiave:valore”, dove è rispettato l’ordine degli elementi (quindi prima la parola chiave, poi i due punti e poi il valore associato).
Questo è un dettaglio importante ai fini di poter creare un exploit con un payload che venga accettato, infatti potremo scrivere un file eseguibile in python dove alla fine di ogni rigo concluderemo con un commento che ci permetterà di rispettare le regole.

Per esempio una stringa del tipo “ print(“test”) #:valore ”.
O ancora più interessante potrebbe essere approfittare di questa vulnerabilità per sovrascrivere il file wazuh-apid.py, ovvero lo script che viene eseguito all’avvio dell’applicazione, essendo un applicativo open-source sarà facile copiare il contenuto del file, aggiungere i commenti alla fine di ogni riga, aggiungere del codice malevolo e quindi ritrovarsi una versione del file wazuh-apid.py che, se eseguito al posto dell’originale comporterebbe l’esecuzione di codice malevolo a nostro favore. Tuttavia se si provasse ad effettuare l’upload del nostro wazuh-apid.py, tramite una richiesta put all’URI /var/ossec/api/scrpits/, si otterrebbe un messaggio d’errore come risposta, perchè effettivamente sul file wazuh-apid.py nel server i permessi di scrittura sono riservati esclusivamente all’utente root , ma il processo dell’API Service viene lanciato dall’utente ossec, motivo per il quale non viene concesso di sovrascrivere il file.
Eppure tornando ad analizzare la documentazione dell’API scopriamo che ci è concesso modificare delle configurazioni, tra le quali l’attributo drop privileges attraverso l’upload di un file .yaml. Portando, quindi, il valore dell’attributo a false e richiedendo un riavvio da remoto dell’applicativo vedremo che il processo relativo alla nostra API adesso viene eseguito dall’utente root e quindi avrà i permessi per sovrascrivere il file wazuh-apid.py con la nostra versione contenente codice malevolo.
Sicuramente ci si sarebbe potuti limitare ad analizzare la documentazione, capire che, attraverso una richiesta put per l’upload del file .yaml con la configurazione a false dell’attributo drop privileges e un successivo riavvio si sarebbe potuto ottenere una privilege escalation sul server eseguendo l’applicativo, e conseguentemente il codice caricato da noi, come utente root. Ma studiare ed analizzare i singoli passaggi dell’analisi fatti da Davide Meacci nel suo script ci permette di capire non solo come creare un exploit per la vulnerabilità ma anche come utilizzarla a nostro vantaggio nel migliore dei modi.

DIFESA

La tecnica di difesa da utilizzare ad oggi consiste semplicemente nel seguire quella che si può definire una “best practice”, ovvero aggiornare l’applicazione all’ultima versione disponibile, o comunque all’ultima patch di sicurezza. Infatti aggiornando semplicemente alla versione 4.0.4 il problema oggi viene facilmente risolto. Ma in questa sede non ci si limiterà solamente a dire quanto semplice sia mettere in sicurezza il nostro applicativo contro questa vulnerabilità, bensì si analizzerà cosa è stato fatto per risolvere il problema. La documentazione ci informa che, con l’aggiornamento alla versione 4.0.4

,viene risolto il problema di path traversal flow, neutralizzando lo sfruttamento di una vulnerabilità generata da un cattivo controllo del path che viene inserito a seguito delle richieste all' API. Nello specifico possiamo andare a visionare i cambiamenti dalla versione 4.0.3 alla 4.0.4 nel repository su github, evidenziando 2 modifiche sostanziali.

1. La funzione is_safe_path effettua un controllo per individuare eventuali sequenze di escape ('./' , '../') per sfruttare un path trasveral attack.
2. L'espressione regolare a cui si fa riferimento per la validazione del path viene valutata una quarta volta, rispetto alle 3 della versione 4.0.3.
3. Le funzionalità per modificare le configurazioni e successivamente riavviare l'applicativo sono state rimosse.

Analizzando i tre punti vediamo come, grazie al controllo sui caratteri di escape e al quarto confronto con una regex, non ci sarà possibile accedere o caricare file in cartelle che non siano previste dalle policy dell'applicazione, mentre ci sarà impossibile effettuare una privilege escalation poichè non potremo modificare le configurazioni e successivamente riavviare l'applicativo per permettere di avviarlo come utente root.

PROCEDURA

Per dimostrare che la difesa di cui sopra è efficace sono state prese in considerazione 3 macchine virtuali collegate ad una rete interna. Due di queste macchine fungeranno da server e avranno avviato i processi di wazuh manager, una nella versione 4.0.3 ed una nella versione 4.0.4. La terza macchina invece verrà utilizzata per effettuare, in ordine, i passaggi descritti nella sezione exploit nel tentativo di riavviare l'applicativo come utente root e sovrascrivere il file wazuh-apid.py.
Dopo diversi tentativi si è dimostrato che è possibile ottenere questo risultato effettuando l'attacco contro la macchina con la versione 4.0.3 del servizio e che, invece, effettuando l'aggiornamento alla 4.0.4 come nella seconda macchina, non vi è nè l'opportunità di effettuare una privilege escalation nè di effettuare la sovrascrittura di file ai quali l'utente ossec non ha permessi di scrittura.
In entrambi i test i passaggi effettuati sono:

1. Effettuare l'accesso all'applicativo ottenendo così un token.
2. Aggiornare la configurazione del server portando drop_privileges a false
3. Riavviare il server
4. Effettuare la sovrascrittura del file wazuh-apid.py

5. Riavviare nuovamente il server

Tutti e 5 questi passaggi sono stati sintetizzati utilizzando uno script in python che è possibile reperire su github al link lasciato nei riferimenti. Lo script è stato leggermente modificato per l'esigenza ma comunque non differisce nelle sue funzionalità principali.
Quindi i test mostreranno come, lanciando lo script di cui si è scritto, la macchina con la versione 4.0.3 del software prima consentirà il riavvio del processo come utente root e successivamente darà la possibilità di sovrascrivere il file wazuh-apid.py.
Mentre il test sulla macchina con il software in versione 4.0.4 non permetterà il riavvio dell'applicativo, evitando così la privilege escalation ma restituirà anche un errore nel momento in cui si tenterà di sovrascrivere il file wazuh-apid.py.

RISULTATI

I risultati dei test dimostrano come la versione 4.0.3 soffra di una serie di problematiche quali: vulnerabilità al path trasversal attack e alla privilege escalation ed inconsistenza delle policy. Infatti si è potuto osservare come eseguendo lo script il servizio sulla macchina server viene in un primo momento riavviato, questa volta come utente root, e successivamente ci viene dato il permesso di sovrascrivere il file wazuh-apid.py che verrebbe eseguito al successivo riavvio della macchina.
Invece, applicando lo stesso test alla macchina server con il servizio aggiornato gli stessi passaggi verrebbero interrotti in più punti. Di fatti, il servizio in versione 4.0.4, alla richiesta PUT per inserire una nuova configurazione non si comporterà come richiesto, successivamente al riavvio, noteremo che l'applicativo si è riavviato nuovamente come utente ossec e ai nostri tentativi di effettuare un path trasversal attack ci restituirà una risposta d'errore. Infine ci verrà negata la possibilità di sovrascrivere il file wazuh-apid.py in quanto non avremo i permessi necessari.

INTERPREATZIONI PERSONALI

Come abbiamo visto questa vulnerabilità può tramutarsi velocemente in un reale problema nel momento in cui uniti i punti analizzati nel paragrafo exploit si riesce a sfruttare per uno scopo reale ben definito, come ad esempio la possibilità di sovrascrivere il file wazuh-apid.py. Andando ad effettuare un attacco del genere si andrà

a minare l'integrità dell'applicativo e delle sue funzionalità, verrà meno il sistema di autenticazione in quanto come visto in fase di analisi sarebbe possibile ottenere il contenuto del jwt_secret, dandoci la possibilità di auto-forgiarci un token, ed infine, visto la possibilità di accedere ad URI a cui la nostra persona non dovrebbe avere accesso, anche la proprietà di segretezza cadrebbe insieme alle policy RBAC del software. Quindi una vulnerabilità di questo tipo risulta potenzialmente pericolosa per tutte le proprietà principali di sicurezza.

RIFERIMENTI

1. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-26814
2. https://documentation.wazuh.com/current/
3. https://documentation.wazuh.com/4.0/user-manual/api/reference.html
4. https://github.com/WickdDavid/CVE-2021-26814
5. https://blog.cys4.com/exploit/wazuh/2021/05/16/CVE-2021-26814.html